## 图像修复inpainting

在实际应用中，我们的图像常常会被噪声腐蚀，这些噪声或是镜头上的灰尘或水滴，或是旧照片的划痕，或者是图像遭到人为的涂画（比如马赛克）或者图像的部分本身已经损坏。

图像通常伴随干扰噪声的存在，在某些场景下会出现局部区域图像污染的情形，因此需要对其进行修复。



数字修复可用来解决图像中小区域污染以重构原始图像，近年来被广泛应用在去除文本和Logo图标等场景中，利用重构去除图像中的划痕和污点。

常用的图像修复基于快速行进的修复算法（FMM）的原理思想是先处理待修复区域边缘上的像素点，然后扩大区域进行遍历，直到修复完所有待修复的图像像素。

图像修复技术的原理是什么呢？简而言之，就是利用那些已经被破坏的区域的边缘，即边缘的颜色和结构，根据这些图像留下的信息去推断被破坏的信息区的信息内容，然后对破坏区进行填补 ，以达到图像修补的目的。

如果我们想让这些受到破坏的额图片尽可能恢复到原样，Opencv能帮我们做到吗？OpenCV真的有这个妙手回春的功能！别以为图像修补的工作只能用PS或者美图秀秀那些软件去做，其实由程序员自己写代码去做更加高效！

OpenCV中就是利用inpaint()这个函数来实现修复功能的。

## Inpaint函数功能及参数

void inpaint( InputArray src, InputArray inpaintMask,OutputArray dst, double inpaintRadius, int flags );

通过蒙版图像（提取的污损部分图像），通过算法使用周围像素填充，实现图像修复操作。

第一个参数src，输入的单通道或三通道图像；需要修补的原图像

第二个参数inpaintMask，图像的掩码，单通道图像，大小跟原图像一致，inpaintMask图像上除了需要修复的部分之外其他部分的像素值全部为0；【就是原图像要修复的部分，必须和要修复的图像一样大】

函数实现关键是图像掩码的确定，可以通过阈值筛选或者手工选定。

第三个参数dst，输出的经过修复的图像；

第四个参数inpaintRadius，修补半径；****用于设定像素点修复邻域半径**** ，用于计算当前像素点的差值；

第五个参数flags，修复算法，有两种：

INPAINT\_NS （基于Navier-Stokes方程的方法，基于流体动力学并使用了偏微分方程）

INPAINT\_TELEA（基于Alexandru Telea方法，基于快速行进算法）

//! the inpainting algorithm

enum

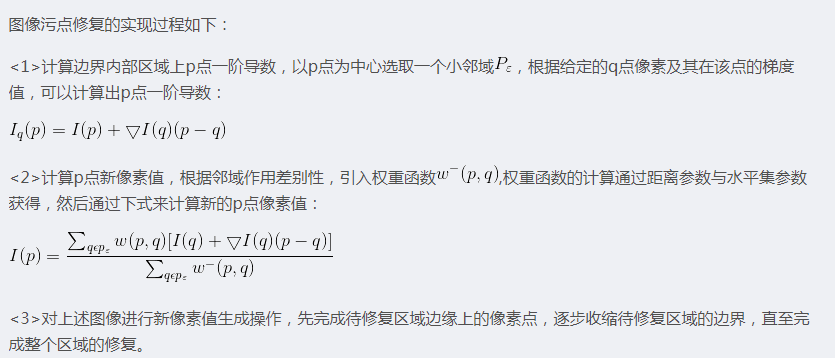
{

INPAINT\_NS = 0, // Navier-Stokes algorithm

INPAINT\_TELEA = 1 // A. Telea algorithm

};





### Inapint源代码

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @Version：OpenCV 3.0.0（Opnencv2和Opnencv3差别不大，Linux和PC的对应版本源码完全一样，均在对应的安装目录下）

\* @源码路径：…\opencv\sources\modules\photo\src\inpaint.cpp

\* @起始行数：810行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void cv::inpaint( InputArray \_src, InputArray \_mask, OutputArray \_dst,

double inpaintRange, int flags )

{

Mat src = \_src.getMat(), mask = \_mask.getMat();

\_dst.create( src.size(), src.type() );

CvMat c\_src = src, c\_mask = mask, c\_dst = \_dst.getMat();

cvInpaint( &c\_src, &c\_mask, &c\_dst, inpaintRange, flags );

}

可以看出inpaint源码中调用的是cvInpaint()函数，其源代码如下。

### cvInpaint()函数，源代码

/\*【inpaint（）源代码】\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @Version：OpenCV 3.0.0（Opnencv2和Opnencv3差别不大，Linux和PC的对应版本源码完全一样，均在对应的安装目录下）

\* @源码路径：…\opencv\sources\modules\photo\src\inpaint.cpp

\* @起始行数：727行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Void cvInpaint( const CvArr\* \_input\_img, const CvArr\* \_inpaint\_mask, CvArr\* \_output\_img,

double inpaintRange, int flags )

{

cv::Ptr<CvMat> mask, band, f, t, out;

cv::Ptr<CvPriorityQueueFloat> Heap, Out;

cv::Ptr<IplConvKernel> el\_cross, el\_range;

CvMat input\_hdr, mask\_hdr, output\_hdr;

CvMat\* input\_img, \*inpaint\_mask, \*output\_img;

int range=cvRound(inpaintRange);

int erows, ecols;

input\_img = cvGetMat( \_input\_img, &input\_hdr );

inpaint\_mask = cvGetMat( \_inpaint\_mask, &mask\_hdr );

output\_img = cvGetMat( \_output\_img, &output\_hdr );

if( !CV\_ARE\_SIZES\_EQ(input\_img,output\_img) || !CV\_ARE\_SIZES\_EQ(input\_img,inpaint\_mask))

CV\_Error( CV\_StsUnmatchedSizes, "All the input and output images must have the same size" );

if( (CV\_MAT\_TYPE(input\_img->type) != CV\_8UC1 &&

CV\_MAT\_TYPE(input\_img->type) != CV\_8UC3) ||

!CV\_ARE\_TYPES\_EQ(input\_img,output\_img) )

CV\_Error( CV\_StsUnsupportedFormat,

"Only 8-bit 1-channel and 3-channel input/output images are supported" );

if( CV\_MAT\_TYPE(inpaint\_mask->type) != CV\_8UC1 )

CV\_Error( CV\_StsUnsupportedFormat, "The mask must be 8-bit 1-channel image" );

range = MAX(range,1);

range = MIN(range,100);

ecols = input\_img->cols + 2;

erows = input\_img->rows + 2;

f.reset(cvCreateMat(erows, ecols, CV\_8UC1));

t.reset(cvCreateMat(erows, ecols, CV\_32FC1));

band.reset(cvCreateMat(erows, ecols, CV\_8UC1));

mask.reset(cvCreateMat(erows, ecols, CV\_8UC1));

el\_cross.reset(cvCreateStructuringElementEx(3,3,1,1,CV\_SHAPE\_CROSS,NULL));

cvCopy( input\_img, output\_img );

cvSet(mask,cvScalar(KNOWN,0,0,0));

COPY\_MASK\_BORDER1\_C1(inpaint\_mask,mask,uchar);

SET\_BORDER1\_C1(mask,uchar,0);

cvSet(f,cvScalar(KNOWN,0,0,0));

cvSet(t,cvScalar(1.0e6f,0,0,0));

cvDilate(mask,band,el\_cross,1); // image with narrow band

Heap=cv::makePtr<CvPriorityQueueFloat>();

if (!Heap->Init(band))

return;

cvSub(band,mask,band,NULL);

SET\_BORDER1\_C1(band,uchar,0);

if (!Heap->Add(band))

return;

cvSet(f,cvScalar(BAND,0,0,0),band);

cvSet(f,cvScalar(INSIDE,0,0,0),mask);

cvSet(t,cvScalar(0,0,0,0),band);

if( flags == cv::INPAINT\_TELEA )

{

out.reset(cvCreateMat(erows, ecols, CV\_8UC1));

el\_range.reset(cvCreateStructuringElementEx(2\*range+1,2\*range+1,

range,range,CV\_SHAPE\_RECT,NULL));

cvDilate(mask,out,el\_range,1);

cvSub(out,mask,out,NULL);

Out=cv::makePtr<CvPriorityQueueFloat>();

if (!Out->Init(out))

return;

if (!Out->Add(band))

return;

cvSub(out,band,out,NULL);

SET\_BORDER1\_C1(out,uchar,0);

icvCalcFMM(out,t,Out,true);

icvTeleaInpaintFMM(mask,t,output\_img,range,Heap);

}

else if (flags == cv::INPAINT\_NS) {

icvNSInpaintFMM(mask,t,output\_img,range,Heap);

} else {

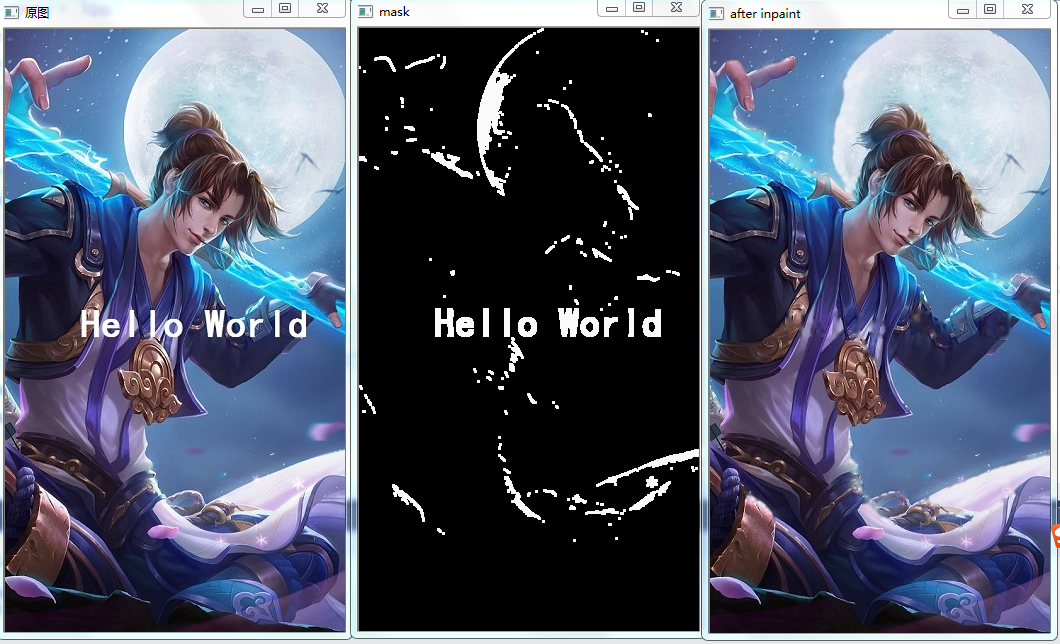
CV\_Error( cv::Error::StsBadArg, "The flags argument must be one of CV\_INPAINT\_TELEA or CV\_INPAINT\_NS" );

}

}

函数实现关键是图像掩码的确定，可以通过阈值筛选或者手工选定，按照这个思路，用二种方法生成掩码，对比图像修复的效果。

## 1.1全区域阈值处理+Mask膨胀处理



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

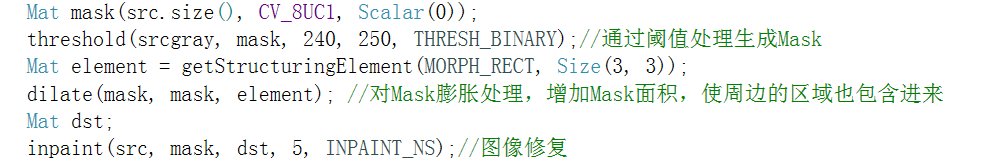
Mat src = imread("e:\\xiake.jpg");

if (!src.data) return -1;

imshow("原图", src);

Mat srcgray;

cvtColor(src,srcgray,COLOR\_BGR2GRAY);

 imshow("mask", mask);

imshow("after inpaint", dst);

waitKey(0);

system("pause");

return 0;

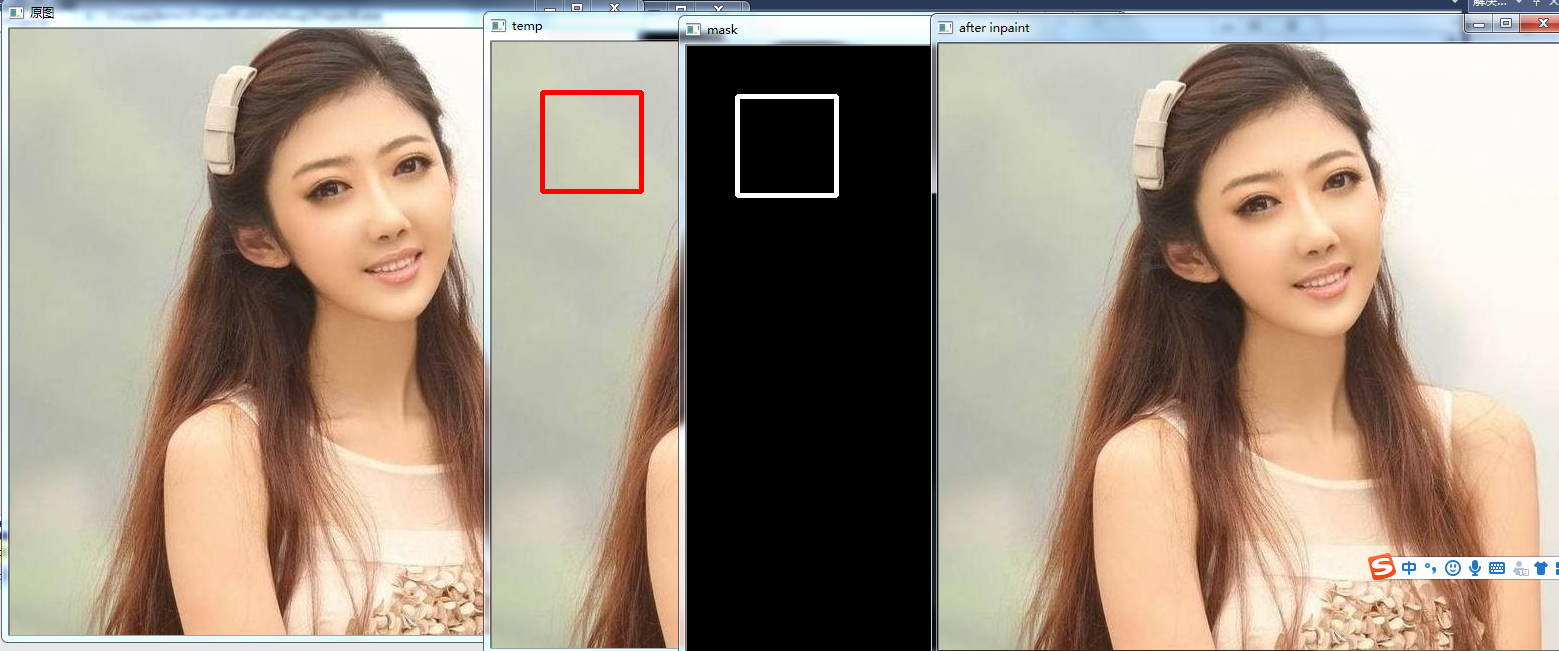
}

受损是由于是图像全区域做阈值处理获得的掩码，图像上部分区域也被当做掩码对待，导致部分图像受损。



无辜的而区域会受损，这个问题能解决一下吗？可以的，那就得自己定义一块需要修复的而区域，不需要修复的区域我们不动它就是了。

## 1.2 inpaint修复污损图像



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat src = imread("e:\\person.jpg");

if (!src.data) return -1;

imshow("原图", src);

Mat dst, mask, temp;

temp = src.clone();

rectangle(temp, Rect(50, 50, 100, 100), Scalar(0, 0, 255), 3, 8);

inRange(temp, Scalar(0, 0, 250), Scalar(0, 0, 255), mask);

inpaint(temp, mask, dst, 3, INPAINT\_TELEA);

imshow("temp", temp);

imshow("mask", mask);

imshow("after inpaint", dst);

waitKey(0);

system("pause");

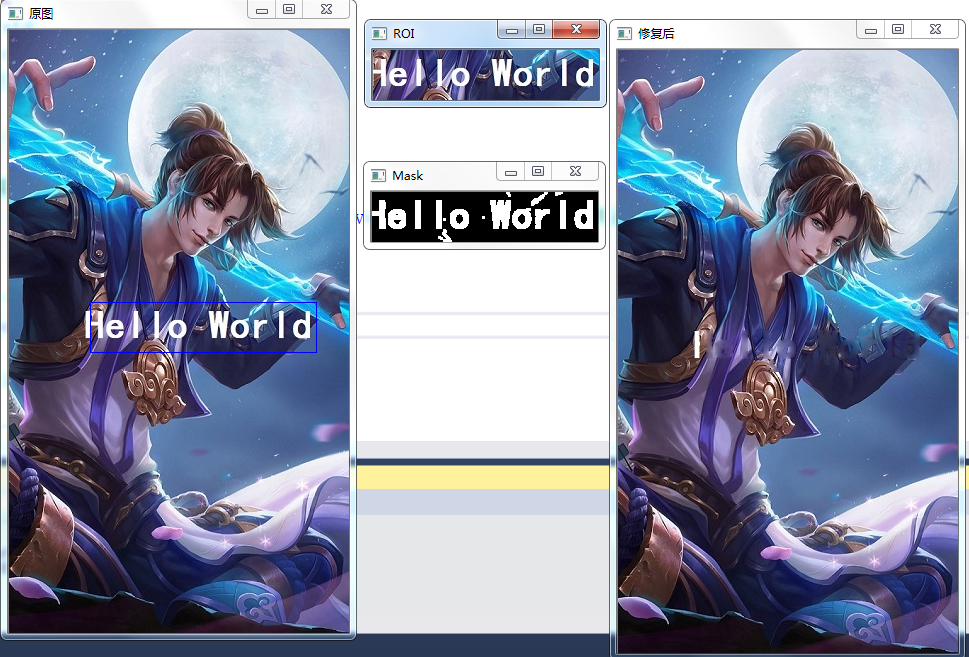
return 0;

}

在原图上画一个矩形，然后找到需要修复位置的掩膜，进而进行图像修复。可以看到，对于简单的图像损坏，修复的还是挺不错的。

## 1.3鼠标圈定区域阈值处理+Mask膨胀处理

下面代码的功能是？



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

/\*\*\*\*\*\*鼠标圈定区域阈值处理+Mask膨胀处理\*\*\*\*\*\*\*/

Point ptL, ptR; //鼠标画出矩形框的起点和终点

Mat imageSource, imageSourceCopy;

Mat ROI; //原图需要修复区域的ROI

void OnMouse(int event, int x, int y, int flag, void \*ustg); //鼠标回调函数

int main()

{

imageSource = imread("e:\\xiake.jpg");

if (!imageSource.data)

{

return -1;

}

imshow("原图", imageSource);

setMouseCallback("原图", OnMouse);

waitKey();

}

void OnMouse(int event, int x, int y, int flag, void \*ustg)

{

if (event == EVENT\_LBUTTONDOWN)

{

ptL = Point(x, y);

ptR = Point(x, y);

}

if (flag == EVENT\_FLAG\_LBUTTON)

{

ptR = Point(x, y);

imageSourceCopy = imageSource.clone();

rectangle(imageSourceCopy, ptL, ptR, Scalar(255, 0, 0));

imshow("原图", imageSourceCopy);

}

if (event == EVENT\_LBUTTONUP)

{

if (ptL != ptR)

{

ROI = imageSource(Rect(ptL, ptR));

imshow("ROI", ROI);

waitKey();

}

}

//单击鼠标右键开始图像修复

if (event == EVENT\_RBUTTONDOWN)

{

imageSourceCopy = ROI.clone();

Mat imageGray;

cvtColor(ROI, imageGray, COLOR\_RGB2GRAY); //转换为灰度图

Mat imageMask = Mat(ROI.size(), CV\_8UC1, Scalar::all(0));

//通过阈值处理生成Mask

threshold(imageGray, imageMask, 235, 255, THRESH\_BINARY);

Mat Kernel = getStructuringElement(MORPH\_RECT, Size(3, 3));

dilate(imageMask, imageMask, Kernel); //对Mask膨胀处理

inpaint(ROI, imageMask, ROI, 9, INPAINT\_TELEA); //图像修复

imshow("Mask", imageMask);

imshow("修复后", imageSource);

}

}

选定区域之外的图像不受修复影响，没有额外的损伤。鼠标划定整个区域作为修复对象，这个方法选定一个矩形区域，把整个矩形区域作为要修复的对象，该方法适用于图像结构比较简单，特别是纯色图像，并且选定区域面积占比不大的情况，效果较好。

## 1.4绘制噪声线条，修复图像

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

/\*\*\*\*\*\*鼠标圈定区域阈值处理+Mask膨胀处理\*\*\*\*\*\*\*/

/\*\*【宏定义部分】\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define WINDOW\_NAME0 "【原始图参考】" //为窗口标题定义的宏

#define WINDOW\_NAME1 "【原始图】" //为窗口标题定义的宏

#define WINDOW\_NAME2 "【修补后的效果图】" //为窗口标题定义的宏

/\*\*【全局变量声明部分】\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Mat srcImage0, srcImage1, inpaintMask;

Point previousPoint(-1, -1);//原来的点坐标

/\*\*【全局函数声明】\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void On\_Mouse(int event, int x, int y, int flags, void\*); //鼠标回调函数

static void ShowHelpText();

int main(int argc, char\*\* argv)

{

system("color 2F");//改变console字体颜色

ShowHelpText(); //显示帮助文字

//载入原始图并进行掩膜的初始化

Mat srcImage = imread("e:\\xiake.jpg", -1);

if (!srcImage.data) { printf("读取图片错误，请确定目录下是否有imread函数指定图片存在~！ \n"); return false; }

srcImage0 = srcImage.clone();

srcImage1 = srcImage.clone();

inpaintMask = Mat::zeros(srcImage1.size(), CV\_8U);

//显示原始图参考

imshow(WINDOW\_NAME0, srcImage0);

//显示原始图

imshow(WINDOW\_NAME1, srcImage1);

//设置鼠标回调消息

setMouseCallback(WINDOW\_NAME1, On\_Mouse, 0);

//轮询按键，根据不同的按键进行处理

while (1)

{

//获取按键键值

char c = (char)waitKey();

if (c == 27) break;//键值为ESC，程序退出

//键值为2，恢复成原始图像

if (c == '2')

{

inpaintMask = Scalar::all(0);

srcImage.copyTo(srcImage1);

imshow(WINDOW\_NAME1, srcImage1);

}

//键值为1或者空格，进行图像修补操作

if (c == '1' || c == ' ')

{

Mat inpaintedImage;

inpaint(srcImage1, inpaintMask, inpaintedImage, 3, INPAINT\_TELEA);

imshow(WINDOW\_NAME2, inpaintedImage);

}

}

return 0;

}

static void On\_Mouse(int event, int x, int y, int flags, void\*)

{

if (event == EVENT\_LBUTTONUP || !(flags & EVENT\_FLAG\_LBUTTON))//鼠标左键弹起消息

previousPoint = Point(-1, -1);

else if (event == EVENT\_LBUTTONDOWN)//鼠标左键按下消息

previousPoint = Point(x, y);

//鼠标按下并移动，进行绘制

else if (event == EVENT\_MOUSEMOVE && (flags & EVENT\_FLAG\_LBUTTON))

{

Point pt(x, y);

if (previousPoint.x < 0)

previousPoint = pt;

//绘制白色线条

line(inpaintMask, previousPoint, pt, Scalar::all(255), 5, 8, 0);

line(srcImage1, previousPoint, pt, Scalar::all(255), 5, 8, 0);

previousPoint = pt;

imshow(WINDOW\_NAME1, srcImage1);

}

}

static void ShowHelpText() // 输出帮助信息

{

//输出欢迎信息和OpenCV版本

printf("\n\n\t\t\t 当前使用的OpenCV版本为：" CV\_VERSION);

printf("\n\n ----------------------------------------------------------------------------\n");

//输出一些帮助信息

printf("\n\n\n\t【图像修复】程序\n");

printf("\n\t请在进行图像修复操作之前，在【原始图】窗口中进行适量的绘制"

"\n\n\t按键操作说明: \n\n"

"\t\t【鼠标左键】-在图像上绘制白色线条\n\n"

"\t\t键盘按键【ESC】- 退出程序\n\n"

"\t\t键盘按键【1】或【SPACE】-进行图像修复操作 \n\n");

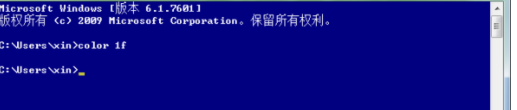
}

图像修复技术在一些简单，颜色单调的图像上进行修复得到的而效果是相当好的，而在一些细节或者复杂的部分进行修复，得到的复原图像的效果就比较一般了。比如在一些背景部分进行修复效果都不错，而在边缘细节上的修复就能看出问题了！

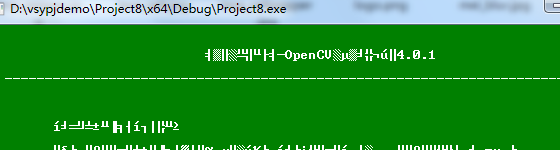
## 补充知识：color2f颜色

C语言中的 system 就是调用dos命令;

color 1f 是用来设置Dos窗口颜色的；前面的 1 设置窗口背景颜色，f 设置窗口字体颜色（前景颜色）；这样的颜色值只有4位，依次是： 高亮，R, G, B 来分的；因此 1 是蓝色，F 是高亮白色。背景蓝色，字体高亮白色;



color2f效果如下



## 补充知识：inRange函数

void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst);

利用inRange函数，通过调节图像颜色信息（H）、饱和度（S）、亮度（V）区间选择我们需要的图像区域

Mat src = imread("e:\\person.jpg");

if (!src.data) return -1;

imshow("原图", src);

Mat dst,mask;

inRange(src, Scalar(235, 235, 235), Scalar(255, 255, 255), mask);

，这个函数就是判断src中每一个像素是否在[lowerb，upperb]之间，注意集合的开闭。如果结果为是，那么在dst相应像素位置填上255，反之则是0。一般我们把dst当作一个mask来用，如上例所示。

官方的解释为：Checks if array elements lie between the elements of two other arrays.即检查数组元素是否在另外两个数组元素值之间。这里的数组通常也就是矩阵Mat或向量。要特别注意的是：该函数输出的dst是一幅二值化之后的图像。

针对单通道图像

dst(I) = lowerb(I)0 ≤ src(I)0 < upperb(I)0

即，如果一幅灰度图像的某个像素的灰度值在指定的高、低阈值范围之内，则在dst图像中令该像素值为255，否则令其为0，这样就生成了一幅二值化的输出图像。

针对三通道图像

dst(I) = lowerb(I)0 ≤ src(I)0 < upperb(I)0 ∧ lowerb(I)1 ≤ src(I)1 < upperb(I)1 ∧lowerb(I)2 ≤ src(I)2 < upperb(I)2

即，每个通道的像素值都必须在规定的阈值范围内！